Amendments to the Specification:

Please replace paragraph **[0090]** with the following rewritten paragraph:

Each write, or other data-modifying operation, to a chunk is performed to all chunk replicas. Leases may be used to maintain a consistent modification order across replicas. Master 130 may grant a chunk lease to one of chunk servers 120 that stores a replica, which may be called the "primary" replica server. Other chunk servers 120 storing the same replica may be called the "secondary" ~~replicas~~ replica servers. The primary replica server selects a serial order for all modifications to the chunk. The primary replica server may provide this serial order to the secondary ~~replicas~~ replica servers in the form of control signals. All of the secondary ~~replicas~~ replica servers follow this order when applying modifications. This lease mechanism may ensure a global order on all modifications to a chunk. The order may be defined first by the lease grant order on master 130, and within a lease, by the serial numbers assigned by the primary replica server.

Please replace paragraph **[0091]** with the following rewritten paragraph:

The lease mechanism minimizes management overhead of master 130. The lease may have an initial timeout period (e.g., 60 seconds), which may be extendable by the primary replica server. For example, as long as a chunk is being modified, the primary replica server can request and typically receive extensions from master 130 indefinitely. These extension requests and grants may be piggybacked on the heartbeat messages regularly exchanged between master 130 and chunk servers 120. If master 130 loses communication with the primary replica server, it can safely grant a new lease to another replica server after the old lease expires. Master 130 may

sometimes attempt to revoke a lease before it expires (e.g., when master 130 wants to disable modifications to a file that is being renamed).

Please replace paragraph **[0093]** with the following rewritten paragraph:

When a client 110 has data to write, client 110 sends a request to master 130 for the identity of one of chunk servers 120 that holds the current lease for the chunk (i.e., the primary replica server) and the locations of the other ~~replicas~~ replica servers (i.e., the secondary ~~replicas~~ replica servers) (act 1210). If no chunk server 120 currently has a lease, master 130 may grant a lease to one of chunk servers 120 that stores a replica of the chunk. That chunk server 120 would then be the primary replica server and other chunk servers 120 storing a replica of the chunk would be secondary ~~replicas~~ replica servers.

Please replace paragraph **[0094]** with the following rewritten paragraph:

Master 130 may then send a reply to client 110 with the requested information (act 1220). The reply may include the identity of the primary replica server and the locations of the secondary ~~replicas~~ replica servers. Client 110 may cache this information and use it for further modifications involving the chunk. Client 110 need only contact master 130 again when the primary replica server becomes unreachable or replies that it no longer holds a lease.

Please replace paragraph **[0095]** with the following rewritten paragraph:

Client 110 may push the write data to all of the ~~replicas~~ replica servers (act 1230). Client 110 may push the data in any order it wants. The primary and secondary ~~replicas~~ replica servers

may store the data in an internal buffer (or cache) until the data is used or aged out (e.g., expires). To use network bandwidth efficiently, the flow of data being written may be decoupled from the flow of control information. Because inter-switch links may be potential bandwidth bottlenecks, data may be written so as to minimize the use of inter-switch links and high latency links.

Please replace paragraph **[0096]** with the following rewritten paragraph:

Client 110 may send the write data to the replica server that is closest to it in the network topology. The closest replica server may or may not be the primary replica server. As shown in Fig. 13, secondary replica server A is closest to client 110. Secondary replica server A may forward the data to the replica server that is closest to it among the remaining ~~replicas~~ replica servers in the network topology. As shown in Fig. 13, the primary replica server is closest to secondary replica server A. The primary replica server may forward the data to the replica server that is closest to it among the remaining ~~replicas~~ replica servers in the network topology. As shown in Fig. 13, secondary replica server B is closest to the primary replica server. This process may continue until all of the ~~replicas~~ replica servers receive the write data.

Please replace paragraph **[0097]** with the following rewritten paragraph:

If the network topology is simple, distances can be easily estimated by examining Internet protocol (IP) addresses. Therefore, client 110 may choose a linear ordering of the ~~replicas~~ replica servers to construct a data stream. Pushing of the write data may be done in a linear fashion to fully utilize the network bandwidth of each replica server.

Please replace paragraph **[0098]** with the following rewritten paragraph:

Latency may be minimized by pipelining the data transfer over TCP connections. Client 110 may start writing on a stream connected to secondary replica server A. While secondary replica sever A receives data from client 110, it may start forwarding the data to the next replica sever (e.g., the primary replica server). Similarly, while the primary replica server receives data from secondary replica server A, it may begin forwarding the data to secondary replica server B.

Please replace paragraph **[0099]** with the following rewritten paragraph:

Once all of the ~~replicas~~ replica servers have acknowledged receiving the data, client 110 may send a write request to the primary replica server (act 1240). The write request may identify the write data that was previously pushed to all of the ~~replicas~~ replica servers. The primary replica server may validate the write request and then apply the write request to data stored in its local memory in the assigned serial order. The primary replica server may assign consecutive serial numbers to all write requests that it receives, possibly from multiple clients 110.

Please replace paragraph **[00100]** with the following rewritten paragraph:

The primary replica server may forward the write request to all of the secondary ~~replicas~~ replica servers (act 1250). Each of the secondary ~~replicas~~ replica servers may apply the received write requests in the assigned serial number order. The secondary ~~replicas~~ replica servers may then reply to the primary replica server indicating that they have completed the write operation (act 1260).

Please replace paragraph **[00101]** with the following rewritten paragraph:

The primary replica server may send the replies to client 110 (act 1270). The primary replica server may report any errors encountered at any of the ~~replicas~~ replica servers to client 110. When errors occur, the write operation may have succeeded at an arbitrary subset of the ~~replicas~~ replica servers. In this case, the client write request is considered to have failed, and the modified region is left in an undefined state. Client 110 may handle such errors by retrying the failed write operation. The retry operation may attempt to repeat acts 1230 through 1270 before falling back to retry from the beginning of the write operation at act 1210.

Please replace paragraph **[00103]** with the following rewritten paragraph:

Fig. 14 is a flowchart of exemplary processing for performing a record append operation according to an implementation consistent with the principles of the invention. When client 110 wants to perform a record append operation, client 110 may perform acts similar to acts 1210 through 1230, as described above with regard to Fig. 12. Once all of the ~~replicas~~ replica servers have acknowledged receiving the record to be appended, client 110 may send a record append request to the primary replica server for the last chunk in the file.

Please replace paragraph **[00104]** with the following rewritten paragraph:

The primary replica server may receive the record append request and determine whether the record fits into the current chunk replica (acts 1410 and 1420). For example, the primary replica server may determine whether appending the record to the current chunk would cause the chunk to exceed its maximum size (e.g., 64 MB). Append operations may be restricted to be at

most one fourth of the maximum chunk size, so that they are more likely to fit into a chunk without too much fragmentation.

Please replace paragraph **[00105]** with the following rewritten paragraph:

If appending the record would cause the current chunk to exceed its maximum size, the primary replica server may pad the chunk to its maximum size using, for example, a special padding character (act 1430). The primary replica server may instruct the other ~~replicas~~ replica servers to do the same. The primary replica server may then notify client 110 to retry the append operation on the next chunk (act 1440).

Please replace paragraph **[00106]** with the following rewritten paragraph:

If the record fits within the current chunk, the primary replica server may append the record to the chunk (act 1450). The primary replica server may also forward the append request to the secondary ~~replicas~~ replica servers and inform them of the offset at which it wrote the record so that the secondary ~~replicas~~ replica servers can write the data at exactly the same offset used by the primary replica server, even if this requires over-writing some existing data. The primary replica server may then notify client 110 of the assigned offset (act 1460).

Please replace paragraph **[00107]** with the following rewritten paragraph:

If an append operation fails at any of the ~~replicas~~ replica servers, client 110 retries the operation. As a result of the failure, the individual ~~replicas~~ replica servers may contain different data possibly including multiple copies of the same record. Furthermore, partial contents of an

append operation may also be written under some situations. There may be no guarantee that all replicas are bytewise identical. Instead, it may be guaranteed only that the data is written at least once as an atomic unit (i.e., in one contiguous file region). This property follows readily from the simple observation that for the operation to report success, the data must have been written at the same offset on all replicas of some chunk. Furthermore, after this, all replicas are at least as long as the end of a record and, therefore, any future record will be assigned a higher offset (or a different chunk) no matter which replica is associated with the primary replica server.

Please replace paragraph **[00109]** with the following rewritten paragraph:

With the above processing, the primary replica server may simultaneously receive two or more record append requests for the same file. In this case, the primary replica server may serialize the append requests. The particular order in which the append requests are serviced may be programmable. Also, the primary replica server may concurrently process two or more record append operations. For example, the primary replica server may receive a record associated with one append operation, while processing an append request associated with another append operation.